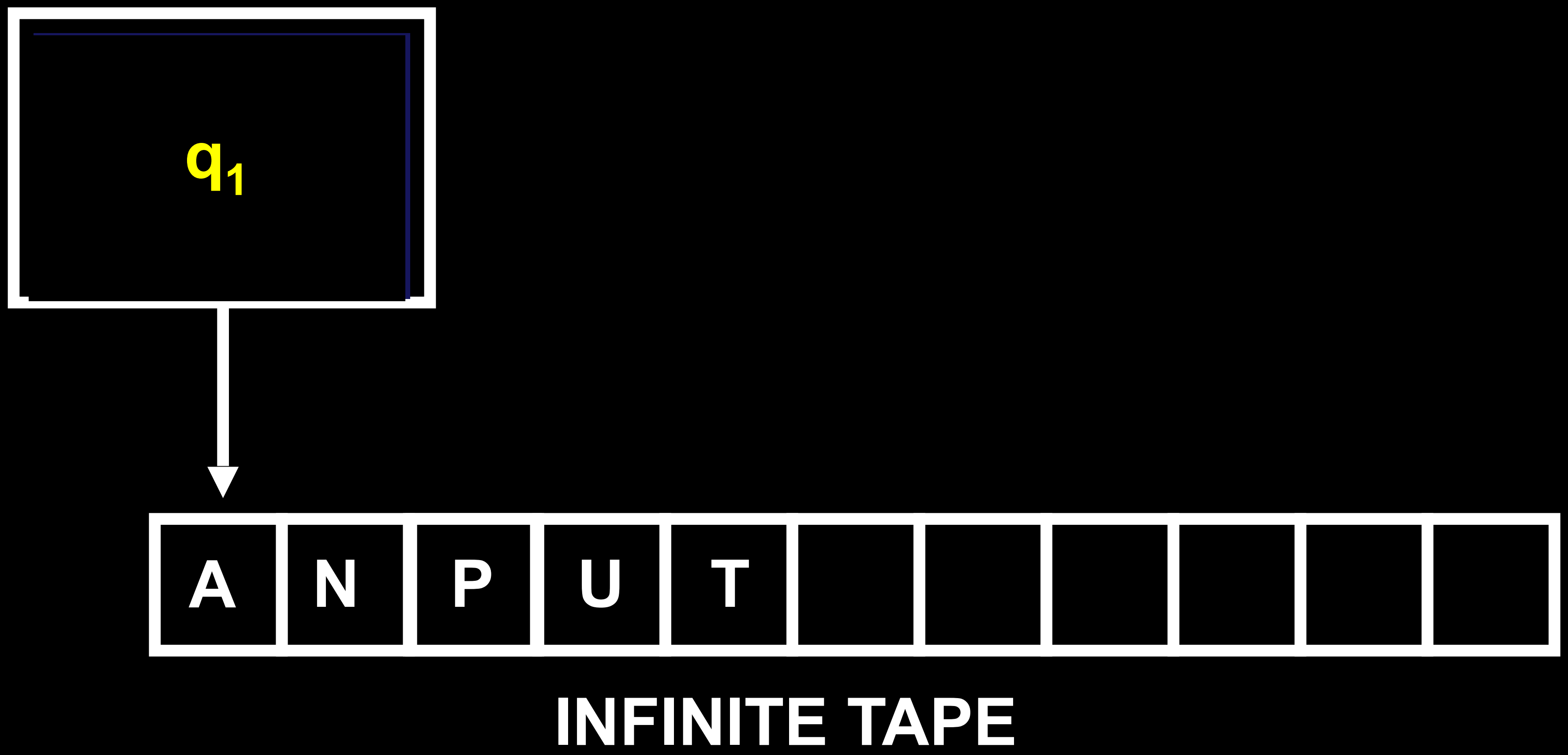
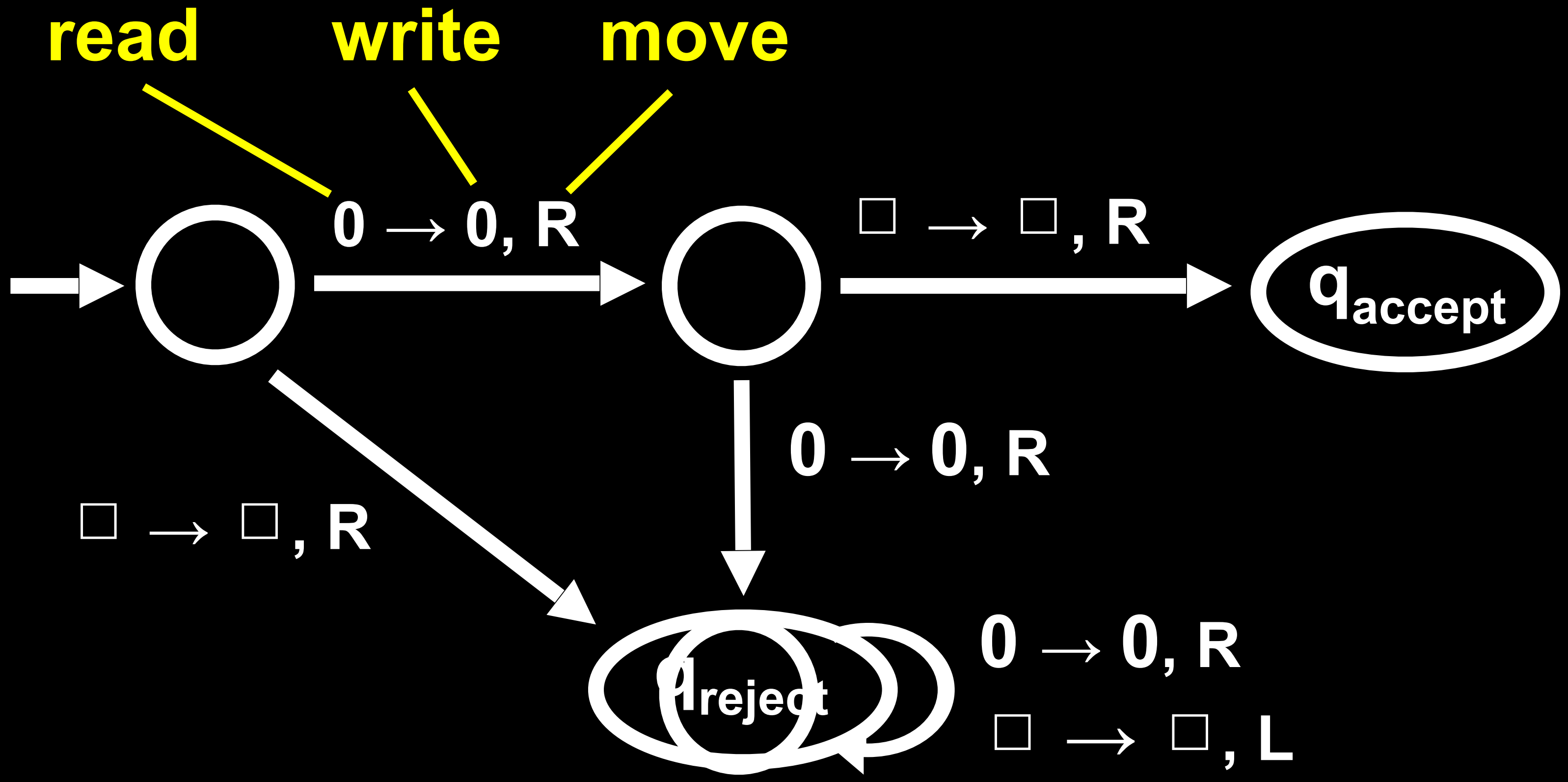
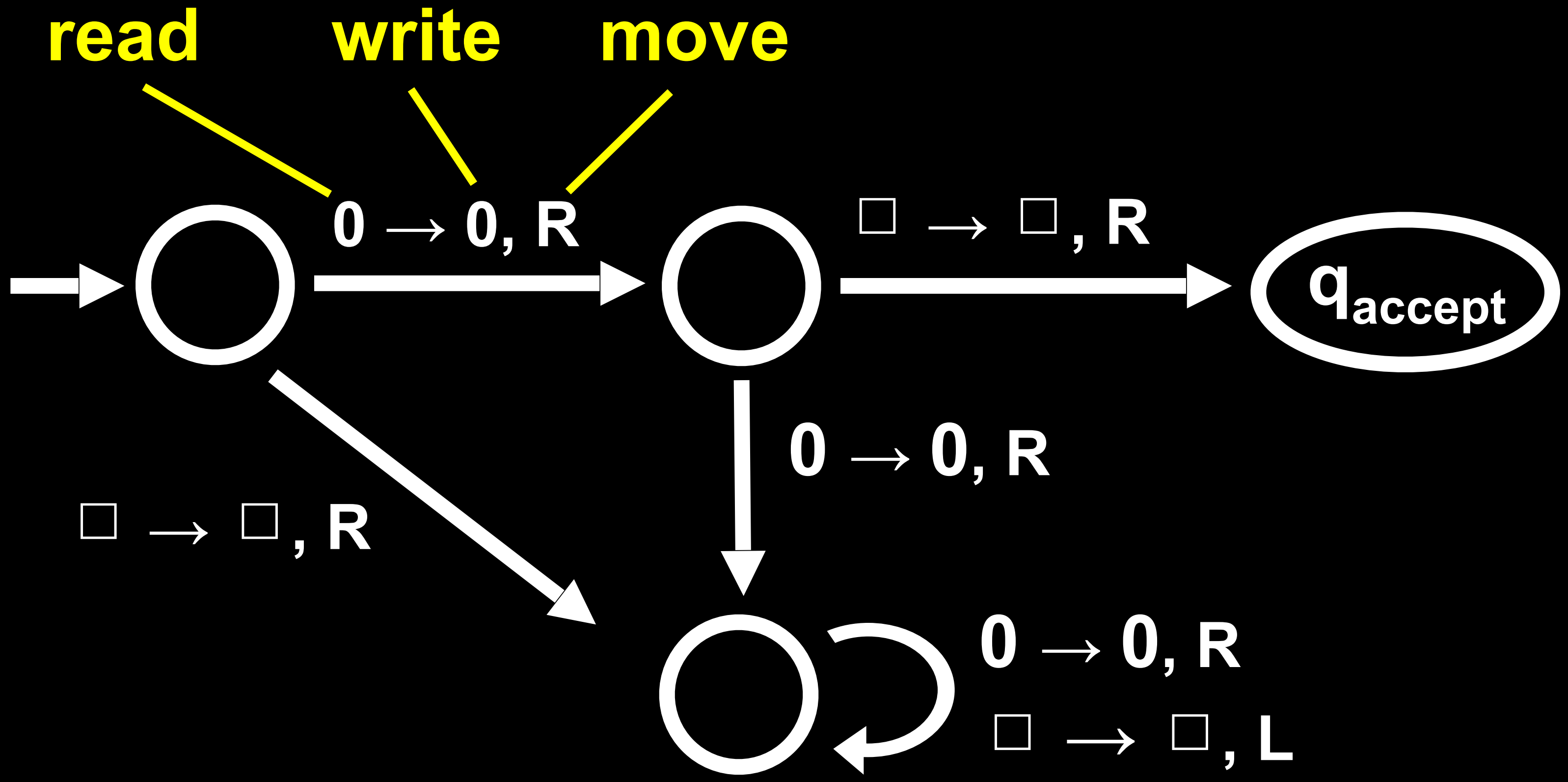
15-453

TURING MAHINES

TURING MACHINE




read
write
move
0 → 0, R
□ → □, R
q_accept
0 → 0, R
□ → □, R
q_reject
0 → 0, R
□ → □, L


read
write
move
0 → 0, R
□ → □, R
q_accept
□ → □, R
0 → 0, R
0 → 0, R
□ → □, L

Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

Q is a finite set of states

Σ is the input alphabet, where $\Box \notin \Sigma$

Γ is the tape alphabet, where $\Box \in \Gamma$ and $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$q_0 \in Q$ is the start state

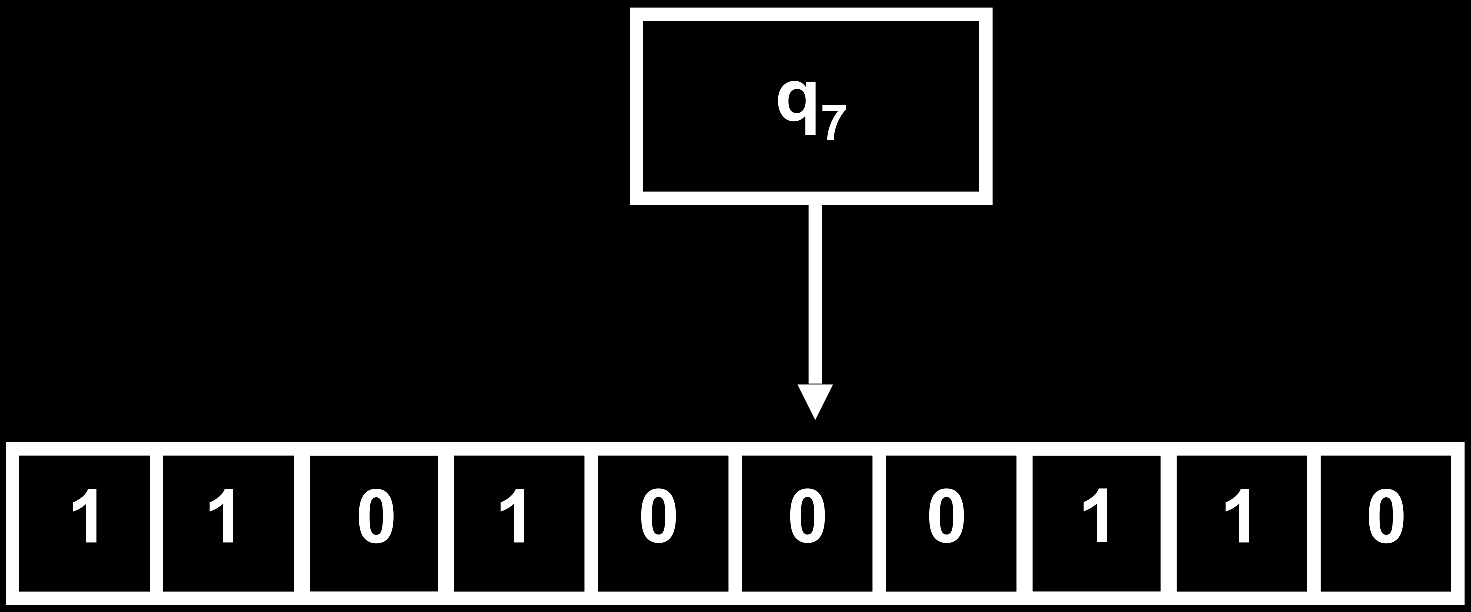
$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

CONFIGURATIONS

$11010q_7 00110$

corresponds to:



A Turing Machine M accepts input w if there is a sequence of configurations $C_1, \dots, C_k$ such that

1. C_1 is a *start* configuration of M on input w, ie

   C_1 is q_0w

2. each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step

$ua\ q_i\ bv$ *yields* $u\ q_j\ acv$ if $\delta(q_i, b) = (q_j, c, L)$

$ua\ qi\ bv$ *yields* $uac\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$

A Turing Machine M accepts input w if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. **C_1 is a *start* configuration of M on input w, ie C_1 is q_0w**
2. **each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step**
3. **C_k is an *accepting* configuration, ie the state of the configuration is q_{accept}**

A Turing Machine M *rejects* input w if there is a sequence of configurations $C_1, \dots, C_k$ such that

1. C_1 is a *start* configuration of M on input w, ie C_1 is q_0w
2. each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step
3. C_k is a *rejecting* configuration, ie the state of the configuration is q_{reject}

A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

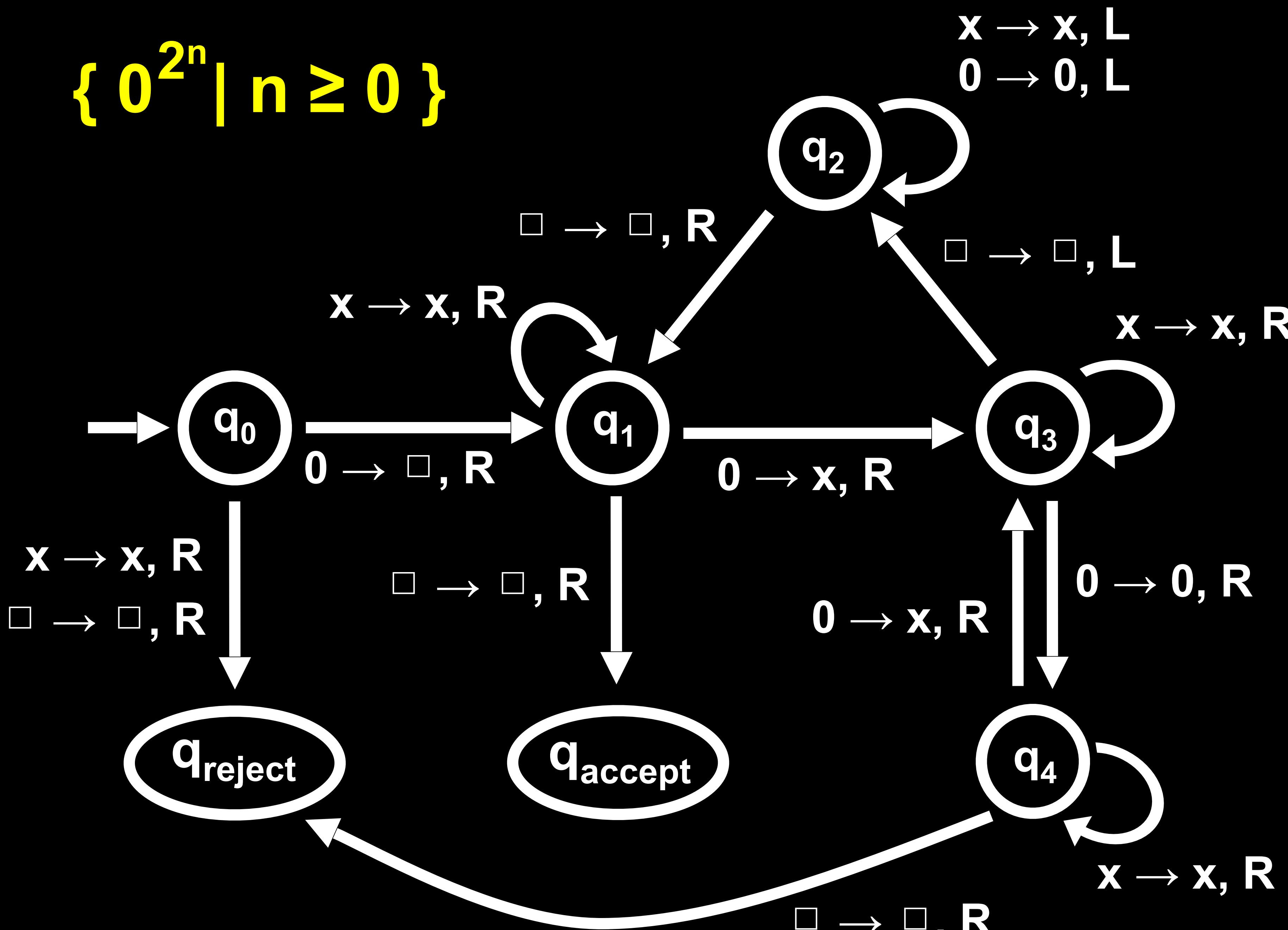
A language is called decidable or recursive if some TM decides it

$\{ 0^{2^n} \mid n \geq 0 \}$ is decidable.

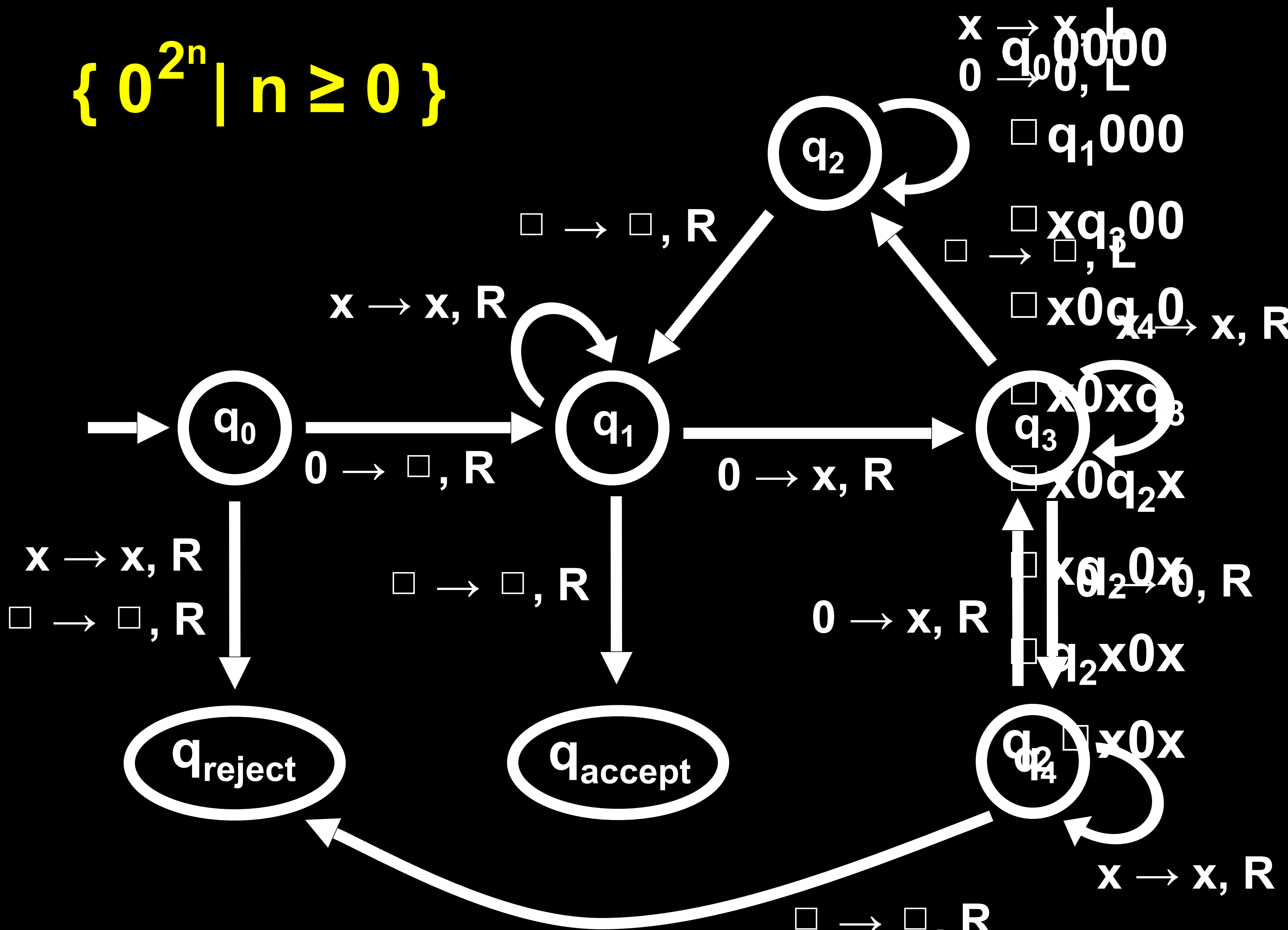
PSEUDOCODE:

1. Sweep from left to right, cross out every other **0**
2. If in stage 1, the tape had only one **0**, *accept*
3. If in stage 1, the tape had an odd number of **0**'s, *reject*
4. Move the head back to the first input symbol.
5. Go to stage 1.

$\{ 0^{2^n} \mid n \geq 0 \}$



$\{ 0^{2^n} \mid n \geq 0 \}$



A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language

A language is called **decidable** or **recursive** if some TM decides it

Theorem: L decidable <-> ¬L decidable

Proof: L has a machine M that accepts or rejects on all inputs. Define M' to be M with accept and reject states swapped. M' decides ¬L.

Theorem: A,B decidable —> A union B decidable

Proof: Let M be a TM for A. Let M' be a TM for B. Make a Union machine implementing the following pseudo-code:
(Intuition: use the even squares to simulate M, and the odd squares to simulate M')

Double input size by writing each input symbol twice starting with q_0 symbols. Use cross product construction to allow the finite state control to remember state of each TM. Move pebble around to always be one square left of position of head in M or M', respectively. Odd phase: Bring head back to start symbol of tape, scan odd squares to find tape head location at pebble… accept if either M or M' accept.

A TM recognizes a language if it accepts all and only those strings in the language

A language is called Turing-recognizable or recursively enumerable, (or r.e. or semi-decidable) if some TM recognizes it

A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

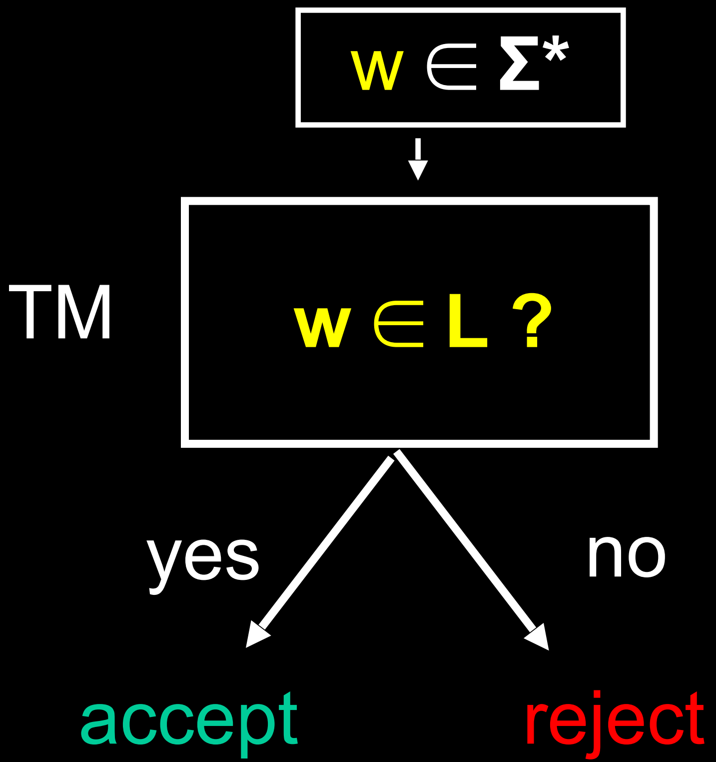
A language is called decidable or recursive if some TM decides it

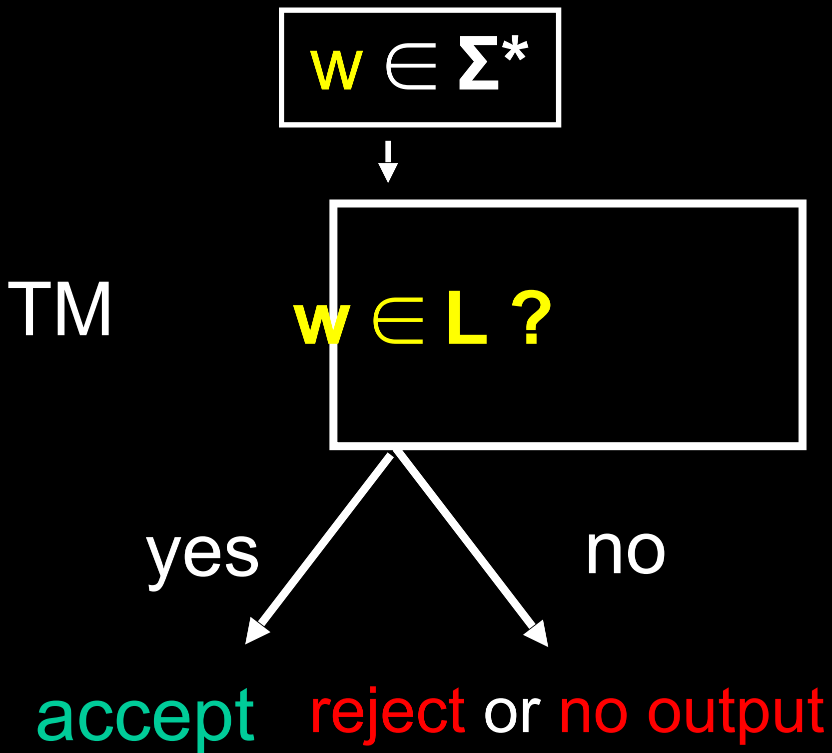
A TM **recognizes** a language if it accepts all and only those strings in the language

A language is called **Turing-recognizable** or **recursively enumerable,** (or **r.e.** or **semi-decidable**) if some TM recognizes it

FALSE: L r.e. <-> ¬L r.e.

Proof: L has a machine M that accepts or rejects on all inputs. Define M' to be M with accept and reject states swapped. M' decides ¬L.



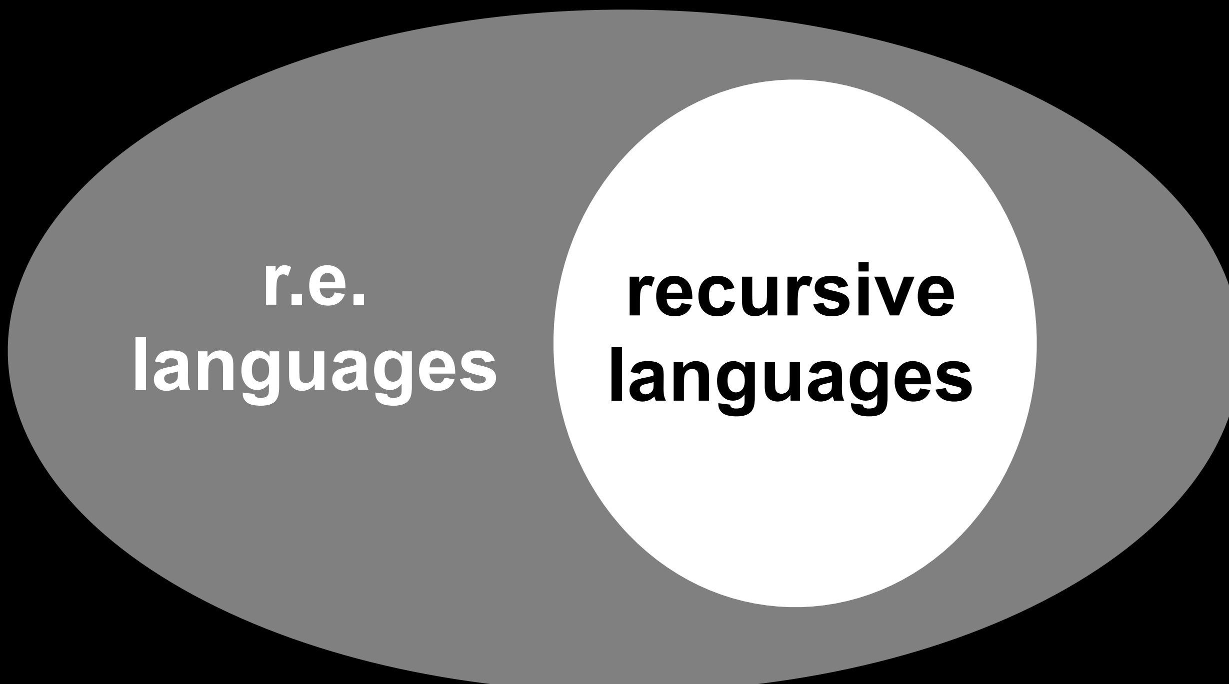


L is decidable
(recursive)

L is semi-decidable
(recursively enumerable,
Turing-recognizable)

A language is called Turing-recognizable or recursively enumerable (r.e.) or semi-decidable if some TM recognizes it

A language is called decidable or recursive if some TM decides it

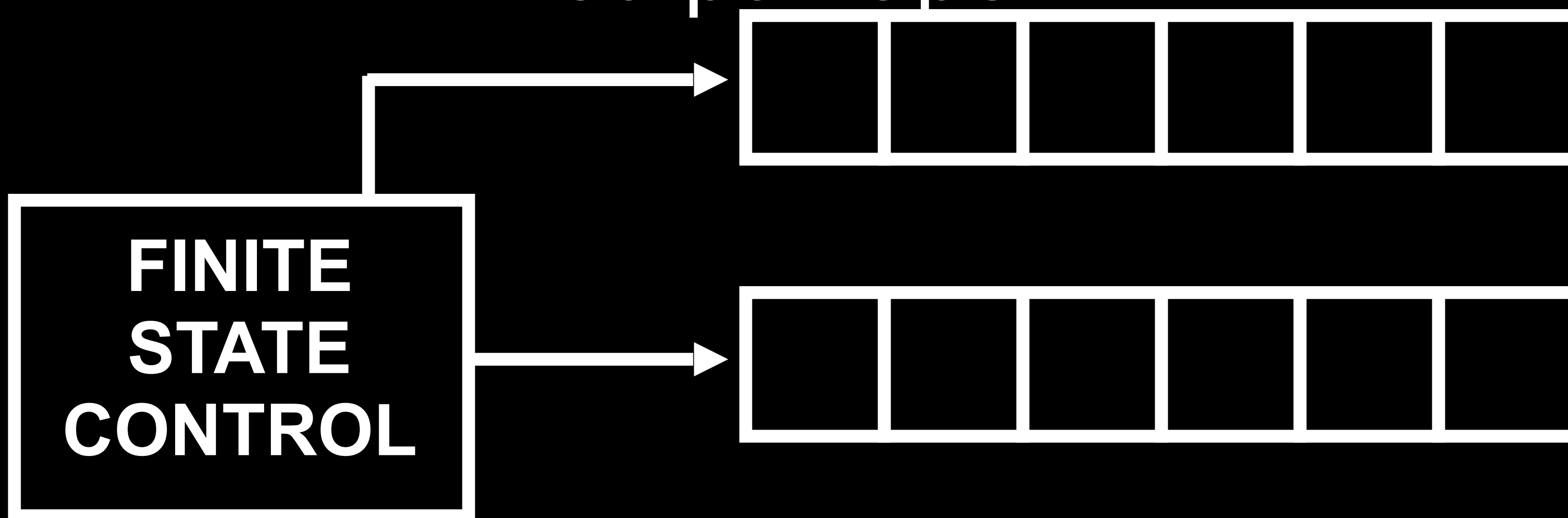


Theorem: If A and ¬A are r.e. then A is recursive

Theorem: If A and ¬A are r.e. then A is recursive

Suppose M accepts A. M' accepts ¬A decidable
Use Odd squares/ Even squares simulation of M and M'. If x is accepted by the even squares reject it/ accepted by the odd squares then accept x.

TURING MACHINE with WRITE ONLY output tape.



Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the sequence of strings occurring on the

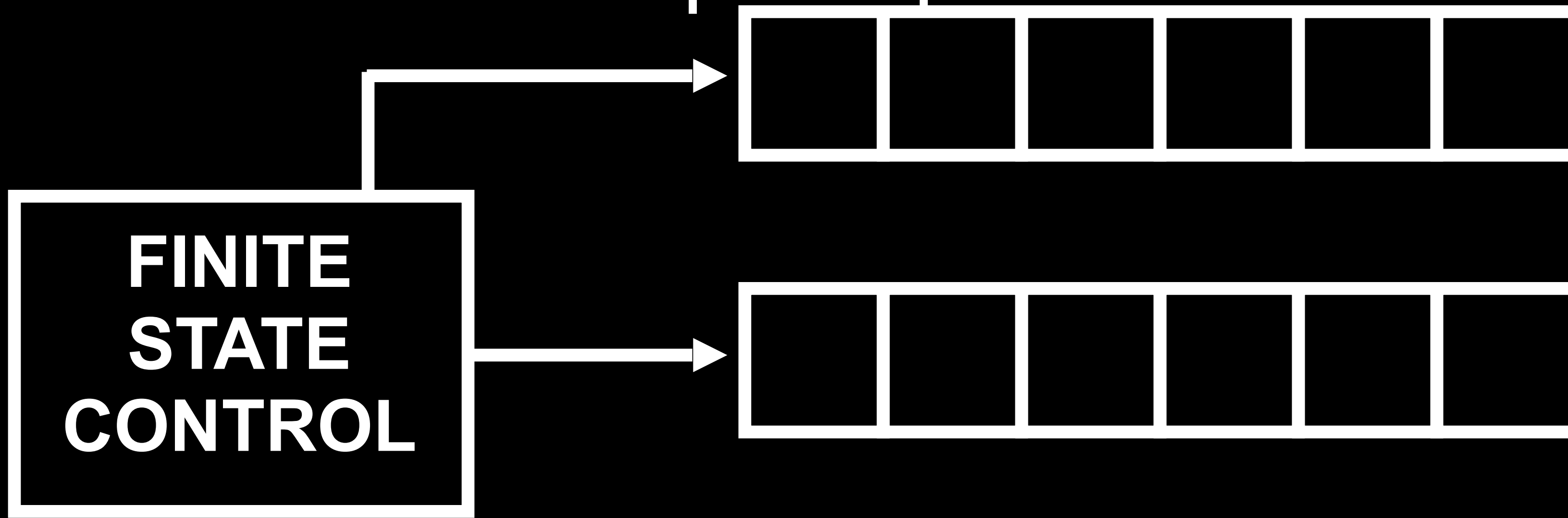
Lex-order has an enumerator strings of length 1, the length 2, ….

Pairs of binary strings have a lex-order enumerator

for each n>0 list all pairs of strings a,b as #a#b# where total length of a and b is n.

Let BINARY(w) = pair of binary strings be any fixed way of encoding a pair of binary strings with a single binary string

TURING MACHINE with WRITE ONLY output tape.



Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the set of strings occurring on the tape.

From every TM M accepting A. there is a TM M' outputting A.

For n = 0 to forever do

{ {Do n parallel simulations of M for n steps for the first n inputs}

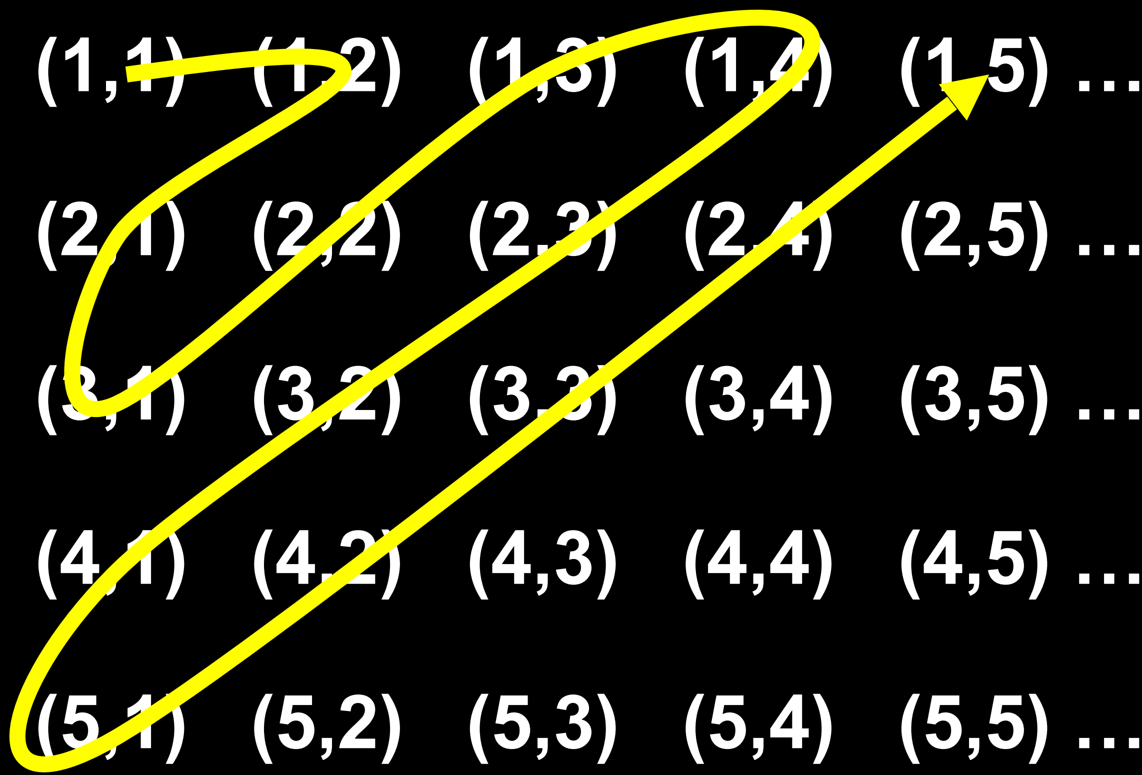
M(0). M(1), M(2), M(3)..

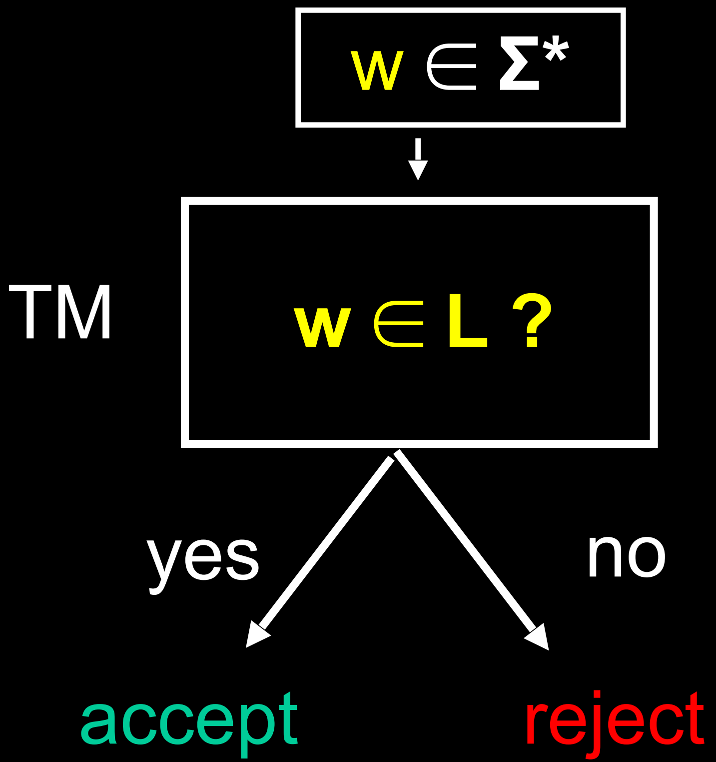
} Odd/Even trick becomes "modulo n" trick. If M(x) accepts then output(x#)

From every TM M outputting A. there is a TM M' accepting A.

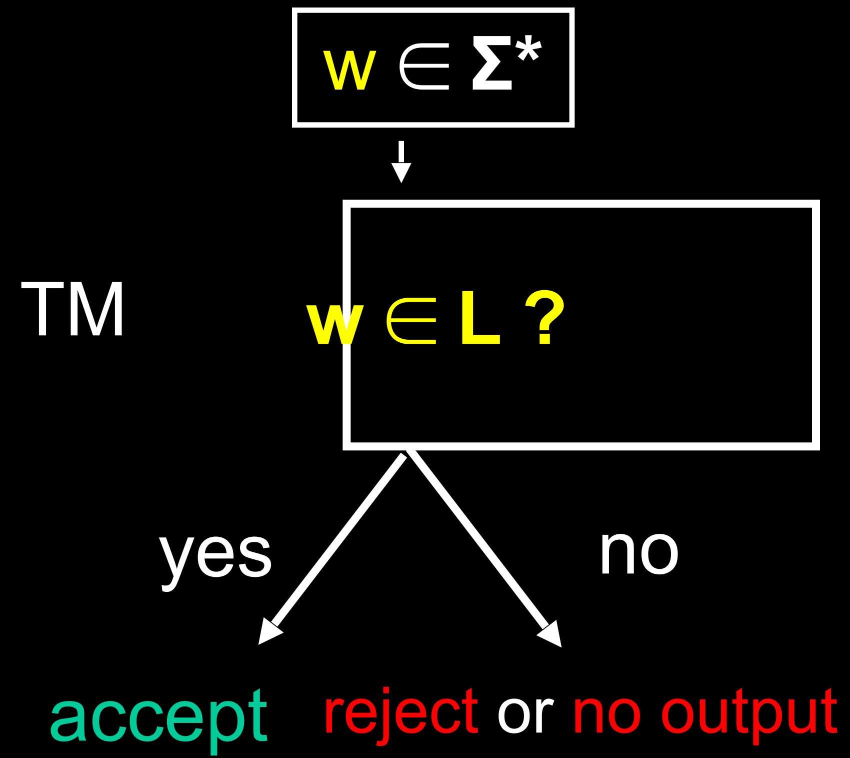
M''(X) run M, accept if X output on tape.

Let $Z^+ = \{1,2,3,4\ldots\}$. There exists a bijection between Z^+ and $Z^+ \times Z^+$ (or Q^+)



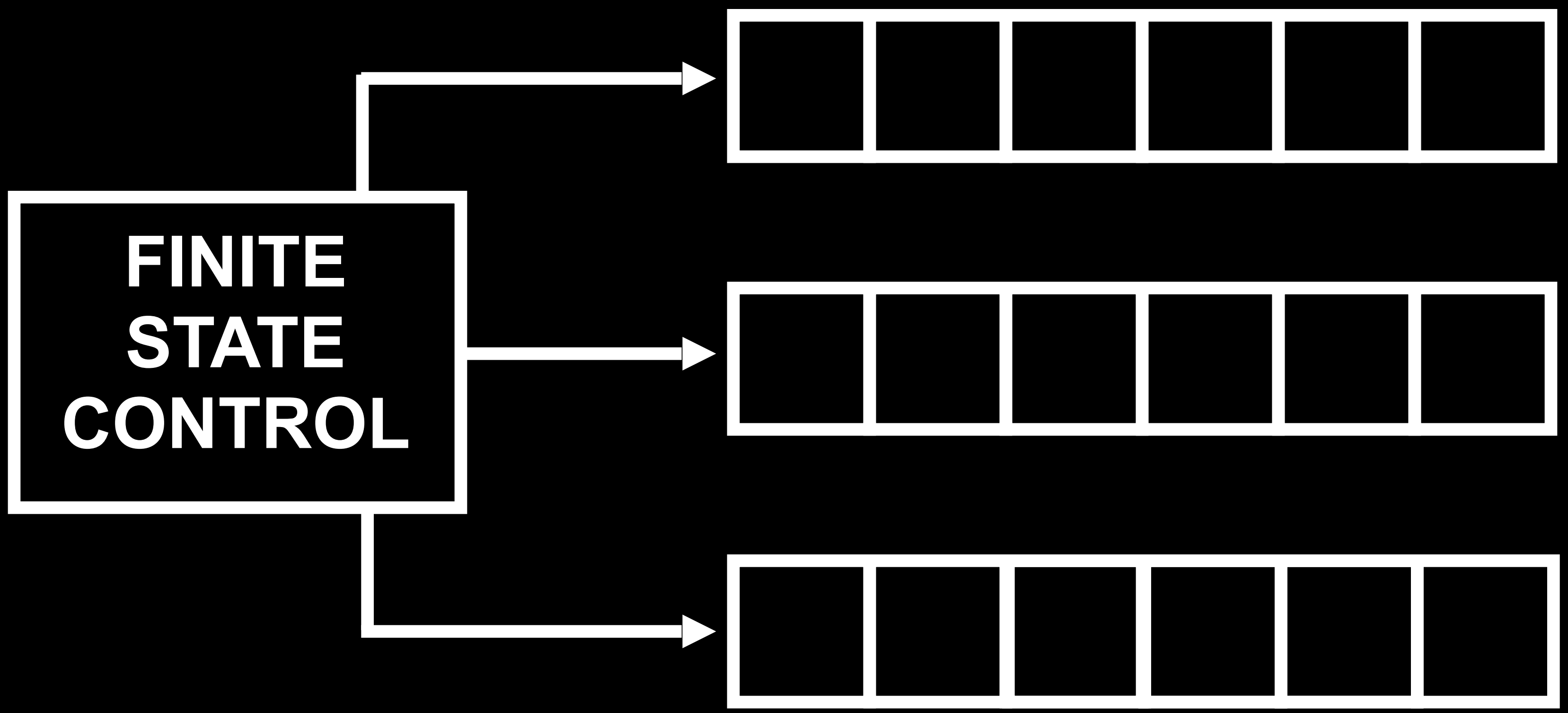


L is decidable
(recursive)



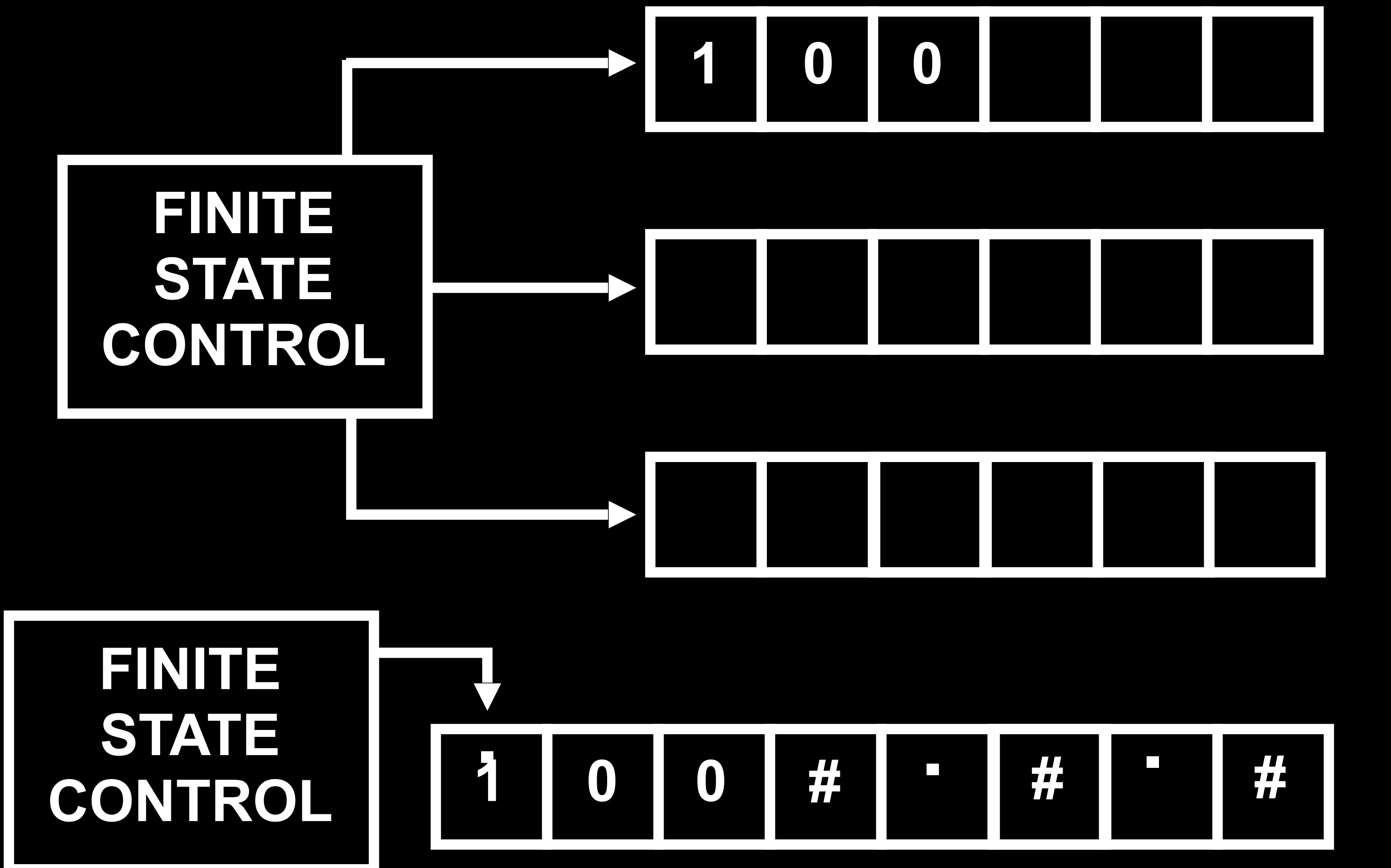
L is semi-decidable
(recursively enumerable,
Turing-recognizable)

MULTITAPE TURING MACHINES

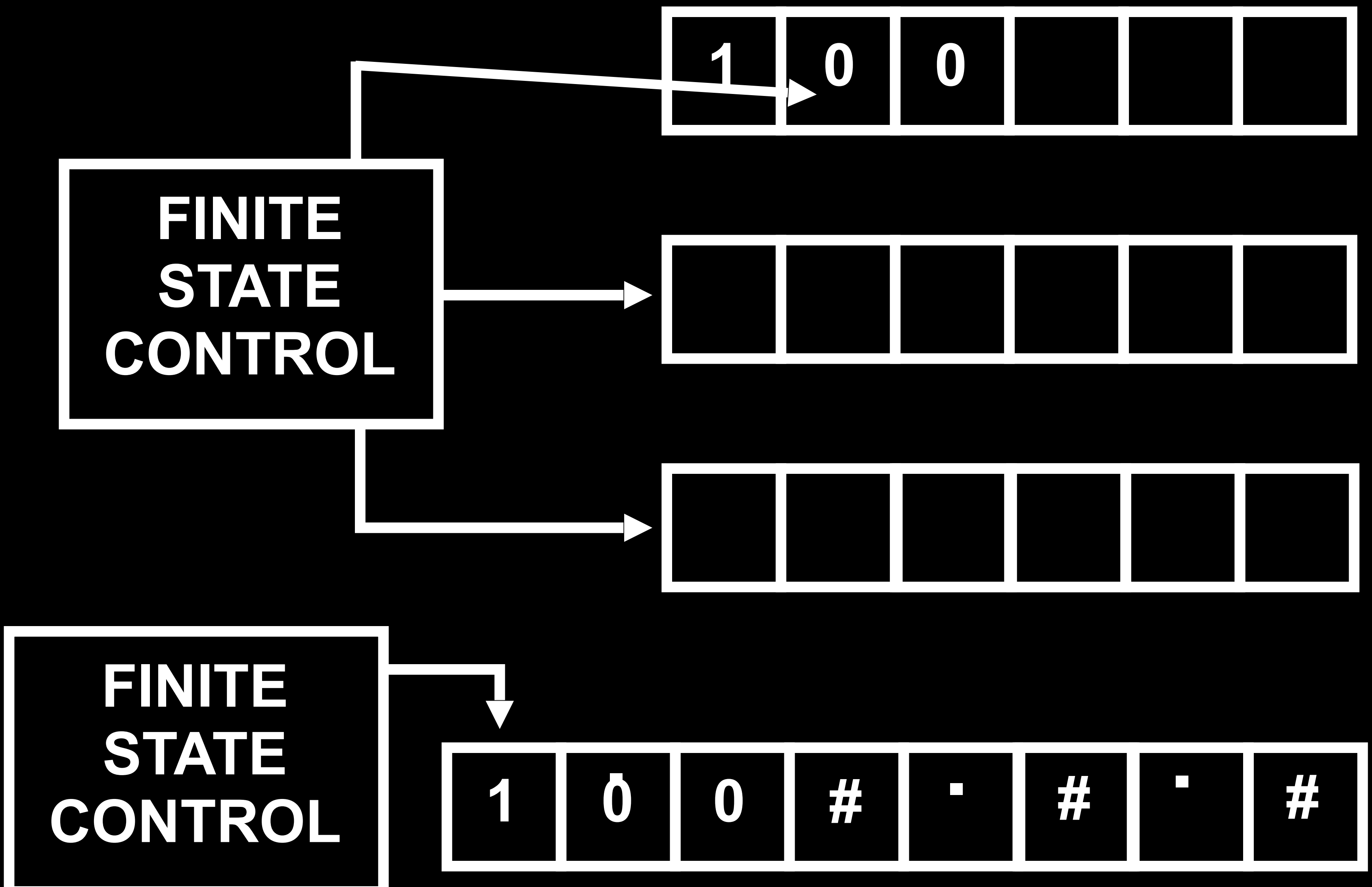


$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



We can encode a TM as a string of 0s and 1s

$$0^n10^m10^k10^s10^t10^r10^u1\ldots$$

- n states
- m tape symbols (first k are input symbols)
- start state
- accept state
- reject state
- blank symbol

$$((p, a), (q, b, L)) = 0^p10^a10^q10^b10$$

$$((p, a), (q, b, R)) = 0^p10^a10^q10^b11$$

THE CHURCH-TURING THESIS

Intuitive Notion of Algorithms
EQUALS
Turing Machines

THE ACCEPTANCE PROBLEM

A_{TM} = { (M, w) | M is a TM that accepts string w }

Theorem: A_{TM} is semi-decidable (r.e.)

but NOT decidable

A_{TM} is r.e. :

Define a TM U as follows:

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

NB. When we write "input (**M**, **w**)" we really mean "input code for (code for **M**, **w**)"

Similarly, we can encode DFAs, NFAs, CFGs, *etc.* into strings of 0s and 1s

So we can define the following languages:

A_{DFA} = { (B, w) | B is a DFA that accepts string w }

Theorem: A_{DFA} is decidable

Proof Idea: Simulate B on w

A_{NFA} = { (B, w) | B is an NFA that accepts string w }

Theorem: A_{NFA} is decidable

A_{CFG} = { (G, w) | G is a CFG that generates string w }

Theorem: A_{CFG} is decidable

Proof Idea: Transform G into Chomsky Normal Form. Try all derivations of length up to $2|w|-1$

UNDECIDABLE PROBLEMS

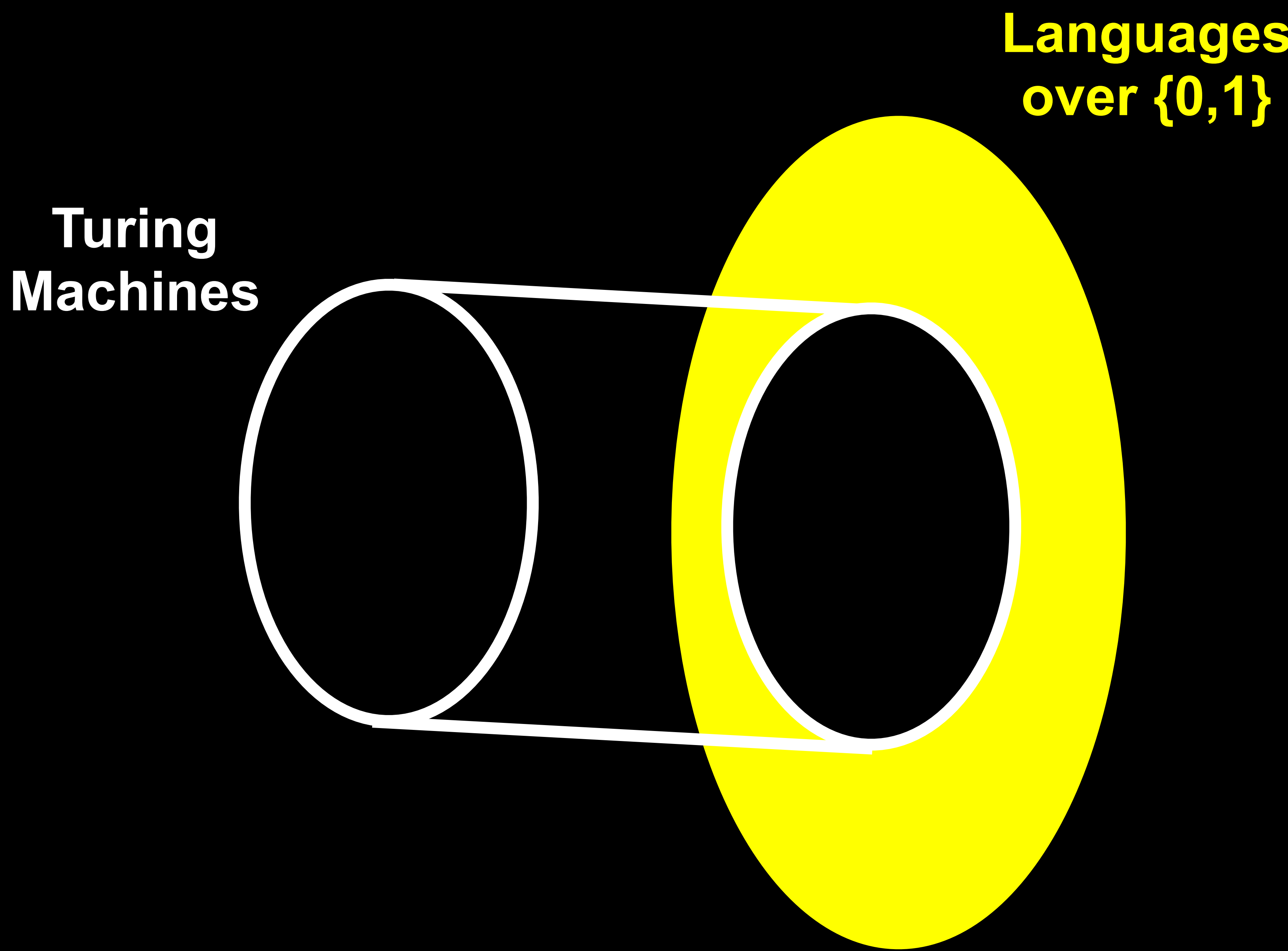
THURSDAY Feb 13

There are languages over {0,1} that are not decidable

If we believe the Church-Turing Thesis, this is **MAJOR**: it means there are things that computers inherently cannot do

We can prove this using a **counting argument**. We will show there is no **onto** function from the set of all Turing Machines to the set of all languages over {0,1}. **(Works for any Σ)** Hence there are languages that have no decider.

Then we will prove something stronger: There are **semi-decidable (r.e.)** languages that are NOT **decidable**


Languages
over {0,1}
Turing
Machines

Let L be any set and 2^L be the power set of L

Theorem: There is no onto map from L to 2^L

Proof: Assume, for a contradiction, that there is an onto map $f : L \rightarrow 2^L$

$$\text{Let } S = \{ x \in L \mid x \notin f(x) \}$$

If $S = f(y)$ then $y \in S$ if and only if $y \notin S$

Can give a more constructive argument!

Theorem: There is no onto function from the positive integers to the real numbers in (0, 1)

Proof: Suppose f is any function mapping the positive integers to the real numbers in (0,

1	→	0.28347279…
2	→	0.88388384…
3	→	0.77635284…
4	→	0.11111111…
5	→	0.12345678…
:		:

2 8 6 1 5

$$[\ n\text{-th digit of } r\ ] = \begin{cases} 1 & \text{if } [\ n\text{-th digit of } f(n)\ ] \neq 1 \\ 2 & \text{otherwise} \end{cases}$$

$f(n) \neq r$ for all n (Here, r = 11121...)

THE MORAL:
No matter what L is,
2^L *always* has more elements than L

Not all languages over {0,1} are decidable, in fact: not all languages over {0,1} are semi-decidable

{decidable languages over {0,1}}

{semi-decidable languages over {0,1}}

{Turing Machines}

{Strings of 0s and 1s}

||

Set L

{Languages over {0,1}}

||

{Sets of strings of 0s and 1s}

||

Set of all subsets of L: 2^L

THE ACCEPTANCE PROBLEM

A_{TM} = { (M, w) | M is a TM that accepts string w }

Theorem: A_{TM} is semi-decidable (r.e.)

but NOT decidable

A_{TM} is r.e. :

Define a TM U as follows:

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

NB. When we write "input (**M**, **w**)" we really mean "input   code for (code for **M**, **w**)"

THE ACCEPTANCE PROBLEM

A_{TM} = { (M, w) | M is a TM that accepts string w }

Theorem: A_{TM} is semi-decidable (r.e.)

but **NOT** decidable

A_{TM} is r.e. :

Define a TM U as follows:

U is a *universal TM*

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

Therefore,

U accepts (M,w) ⇔ M accepts w ⇔ (M,w) ∈ A_{TM}

Therefore, U *recognizes* A_{TM}

A_{TM} = { (**M,w**) | **M** is a TM that accepts string **w** }

A_{TM} **is undecidable:** (proof by contradiction)

Assume machine H decides A_{TM}

H((M,w)) = { Accept if M accepts w ; Reject if M does not accept w }

Construct a new TM D as follows: on input M, run H on (M,M) and output the opposite of H

D(D) = { Reject if D accepts D ; Accept if D does not accept D }

OUTPUT OF H

	M_1	M_2	M_3	M_4 …	D
M_1	accept	accept	accept	reject	accept
M_2	reject	accept	reject	reject	reject
M_3	accept	reject	reject	accept	accept
M_4	accept	reject	reject	reject	accept
:					
D	reject	reject	accept	accept	?

Theorem: A_{TM} is r.e. but NOT decidable

Cor: $\neg A_{TM}$ is not even r.e.!

A_{TM} = { (M,w) | M is a TM that accepts string w }

A_{TM} is undecidable: A constructive proof:

Let machine H semi-decides A_{TM} (Such ∃ , why?)

H((M,w)) = { Accept if M accepts w; Reject or No output if M does not accept w

Construct a new TM D as follows: on input M, run H on (M,M) and output

D(D) = { Reject if H (D, D) Accepts; Accept if H (D, D) Rejects; No output if H D, D) has No output

H((D,D)) = No output

No Contradictions !

We have shown:

Given any machine H for semi-deciding A_{TM}, we can *effectively construct* a TM D such that (D,D) $\notin A_{TM}$ but H fails to tell us that.

That is, H *fails* to be a decider on instance (D,D).

In other words,

Given any "good" candidate for deciding the *Acceptance Problem*, we can effectively construct an instance where the candidate fails.

THE classical HALTING PROBLEM

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

Theorem: $HALT_{TM}$ is undecidable

Proof: Assume, for a contradiction, that TM H decides $HALT_{TM}$

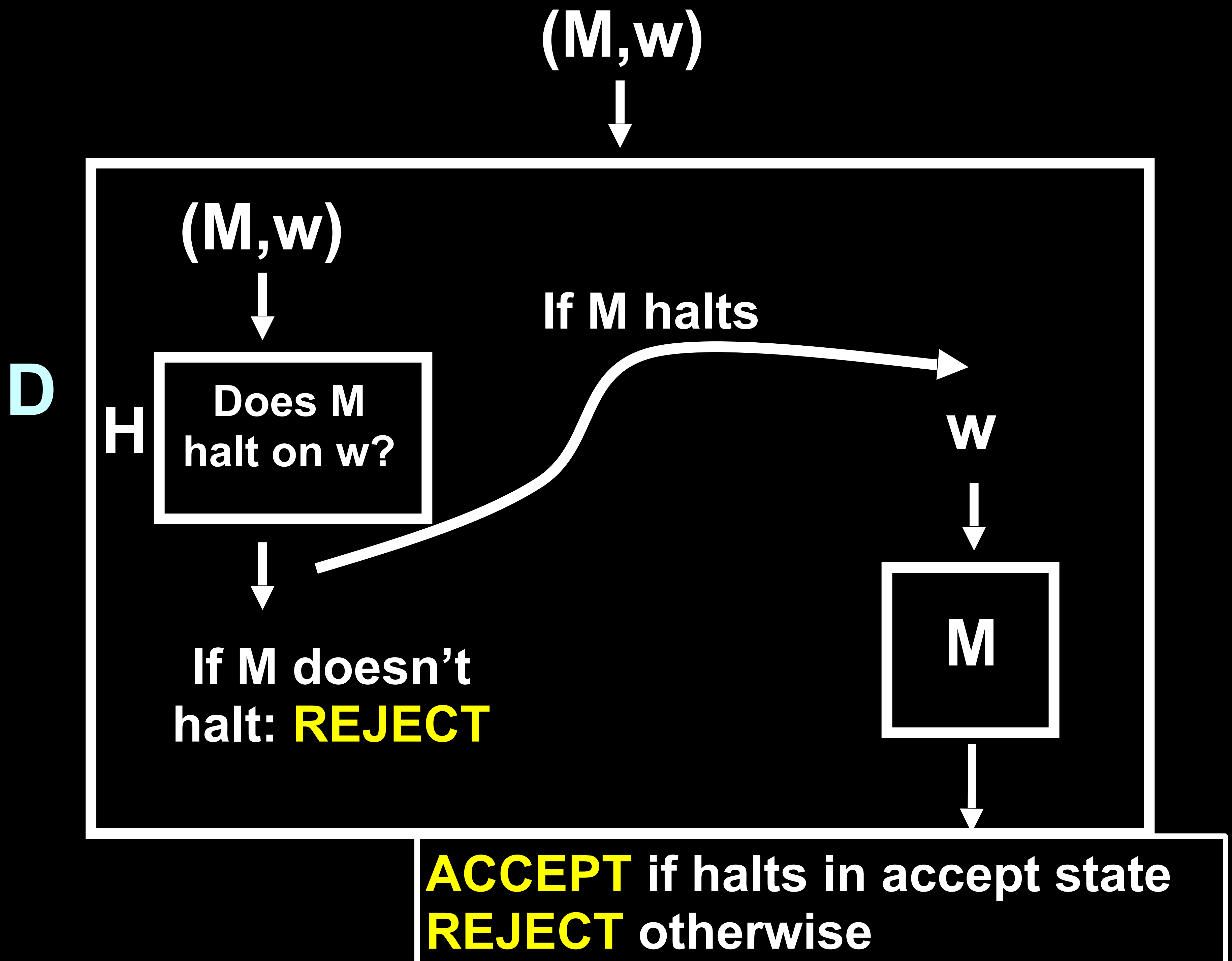
We use H to construct a TM D that decides A_{TM}

On input (M,w), D runs H on (M,w):

If H rejects then reject

If H accepts, run M on w until it halts:

Accept if M accepts, ie halts in an accept state
Otherwise reject


(M,w)
D
(M,w)
H
Does M
halt on w?
If M halts
w
If M doesn't
halt: REJECT
M
ACCEPT if halts in accept state
REJECT otherwise

In many cases, one can show that a language L is undecidable by showing that if it is decidable, then so is A_{TM}

We reduce deciding A_{TM} to deciding the language in question

$$A_{TM} \leq L$$

We just showed: $A_{TM} \leq Halt_{TM}$

Is $Halt_{TM} \leq A_{TM}$?